Tim Kasse

Mr. Tim Kasse co-founded Institute for Software Process Improvement (ISPI) with Jeff Perdue in 1991. Mr. Kasse's accomplishments include the development and delivery of Software Process Improvement seminars and workshops; Software Process Assessment development, customization, training, and coaching; and consulting on the Capability Maturity Model^{SM} and Process Improvement. He is the architect behind the ISPI Action Focused Assessment which has received widespread acceptance throughout Europe. Recently Mr. Kasse has combined the SEI CBA IPI with the critical features from the ISPI Action Focused Assessment Method to create The Enriched CBA IPI.

Prior to forming ISPI, Mr. Kasse spent four years at the Software Engineering Institute. He has participated in over 40 Software Process Assessments (SPAs), 23 of which have been conducted in Europe. During his tenure at the SEI, Mr. Kasse served the Software Process Assessment (SPA) Project as a contributing member, as the task leader for the Self-Assessment effort, and as the Manager of the SPA Project. Mr. Kasse was a major contributor to the development of the Capability Maturity ModelSM, which provides the framework for SEI assessments and evaluations. Mr. Kasse has been certified by the SEI to conduct SEI CBA IPIs. Mr. Kasse is also a trained ISO 9000 auditor. He has 25 years of software related experience.

Tim Kasse Voice: 32-3-605-4875
Chief Executive Officer Fax: 32-3-605-4876
Institute for Software Process Improvement Inc
Email: tckispi@aol.com
Klein Heiken 101
2950 Kapellen
Belgium

Title: Software Configuration Management for Project Leaders
Presenter: Tim Kasse
Track: Track 10
Day: Tuesday
Keywords: Configuration Identification, Baselining, Configuration Control, Status Accounting, Interface Control, Subcontractor Control, Configuration Audits, Software Library, Software Configuration Management Plan, SCM

Abstract: As the systems being built today increase in software content, the need for SCM continues to rise. Prime contractors are integrating millions of lines of code from multiple subcontractors. Companies are required to produce and maintain variants of their main product to reach out to a very diversified market. Project Leaders are aware of the need to better manage and control their projects. This paper brings together most of the Software Configuration Management concepts to be analyzed from the Project Leader point of view. Configuration Management will be shown as one of the most important process improvement tools that a Project Leader can utilize.

Software Configuration Management for Project Leaders

Purpose of Software Configuration Management

According to the Software Engineering Institute's KPA definition of Software Configuration Management, the purpose of Software Configuration Management is to establish and maintain the integrity of the products of the software project throughout the project's software lifecycle. According to Webster, "integrity" is the quality or state of wholeness or completeness. Knowing the state of the product that a project is developing and knowing that it satisfies the customer's requirements is of utmost importance for any Project Leader. Software Configuration Management then can be viewed as a support function that assists a Project Leader to better manage and control the project.

The Need for SCM

In many companies that produce software, software support functions such as Software Quality Assurance and Software Configuration Management are not perceived as value added by Project Leaders and software developers alike. Software Configuration Management is frequently viewed by the developers as a hindrance to the improvement of the product because of the overhead associated with the change control function of Software Configuration Management. But on closer examination, it can be shown that the most frustrating software problems are often caused by poor configuration management.

- The latest version of source code cannot be found.
- A difficult bug that was fixed at great expense suddenly reappears.
- A developed and tested feature is mysteriously missing.
- A fully tested program suddenly does not work.
- The wrong version of the code was tested.
- There is no traceability between the software requirements, documentation, and code.
- Programmers are working on the wrong version of the code.
- The wrong versions of the configuration items are being baselined.
- No one knows which modules comprise the software system delivered to the customer.

While one might expect to find a list such as the one above in a longer list of possibilities published in a book, most of the problems listed above were revealed during software process assessments conducted by this author. Projects, under great stress to meet difficult deadlines, found their scarce time resource constantly under attack due to the rework caused by these reasons. One developer stated that he had “fixed” a problem three times and three months after each delivery the problem reoccurred in the system. Project Leaders cannot afford to have their software developers redoing what was already done.

A key role of Software Configuration Management is to control changes actively in order to answer the following questions. What is the current software configuration? What is the status of my modules? What changes have been made to my software? Do anyone else’s changes affect my software? SCM provides visibility into the status of the evolving software product. SCM answers the Who, What, When, and Why. Who made the changes? What changes were made to the software? When were the changes made? Why were the changes made?

Software Configuration Management versus Change Control

Software Configuration Management is often equated to Change Control. Indeed Change Control is a critical component of Software Configuration Management but it is only one of many. Let us briefly examine the components of Software Configuration Management and connect them to supporting a Project Leader’s ability to manage and control the project.

Configuration Identification

Configuration Identification supports the identifying of the structure of the software system and identifying all of the related life-cycle work products. It provides a unique identifier for each of those work products and it supports traceability between the software and all other related software products.

Two structures that SCM is concerned with directly affect a project:

- Problem solving structure - the system concept evolves through the lifecycle by successive refinement and elaboration of the resulting work products.
- Product system structure - the system is composed of subsystem components, which are themselves composed of subsystem components.

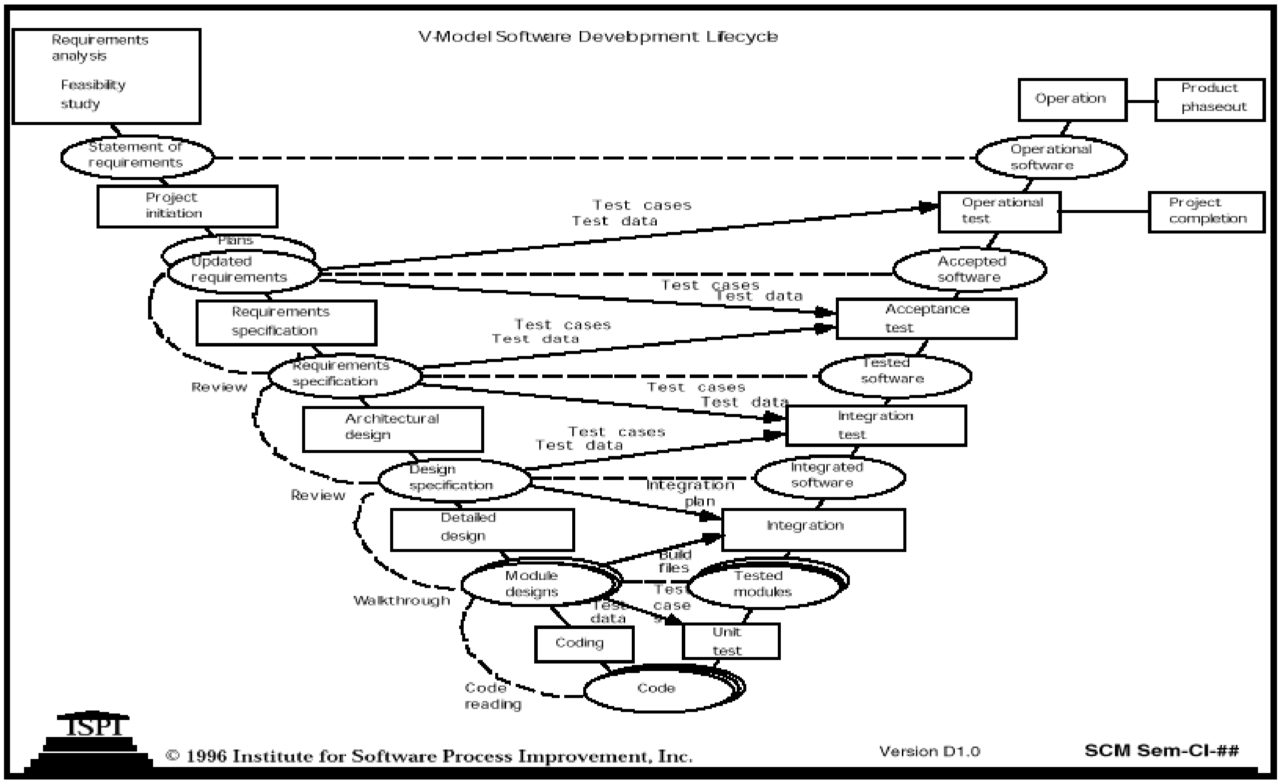

Figure 1. V - Software Lifecycle

Figure 1 shows a V - Software Lifecycle out of which would come a number of predefined work products. Examples of work products that may be identified to be placed under configuration control include:

- Source code modules
- System data files
- System Build Files / Scripts
- Requirements Specification
- Interface Specifications
- Design Specifications
- Software Architecture Specification
- Test Plans
- Test Procedures
- User Documentation
- Software Development Plan
- Quality Plans
- Configuration Management Plans
- Compilers
- Linkers/Loaders
- Debuggers
- Operating Systems
- Shell Scripts
- Third Party Tools
- Other related support tools
- Procedure Language Descriptions
- Development Procedures & Standards

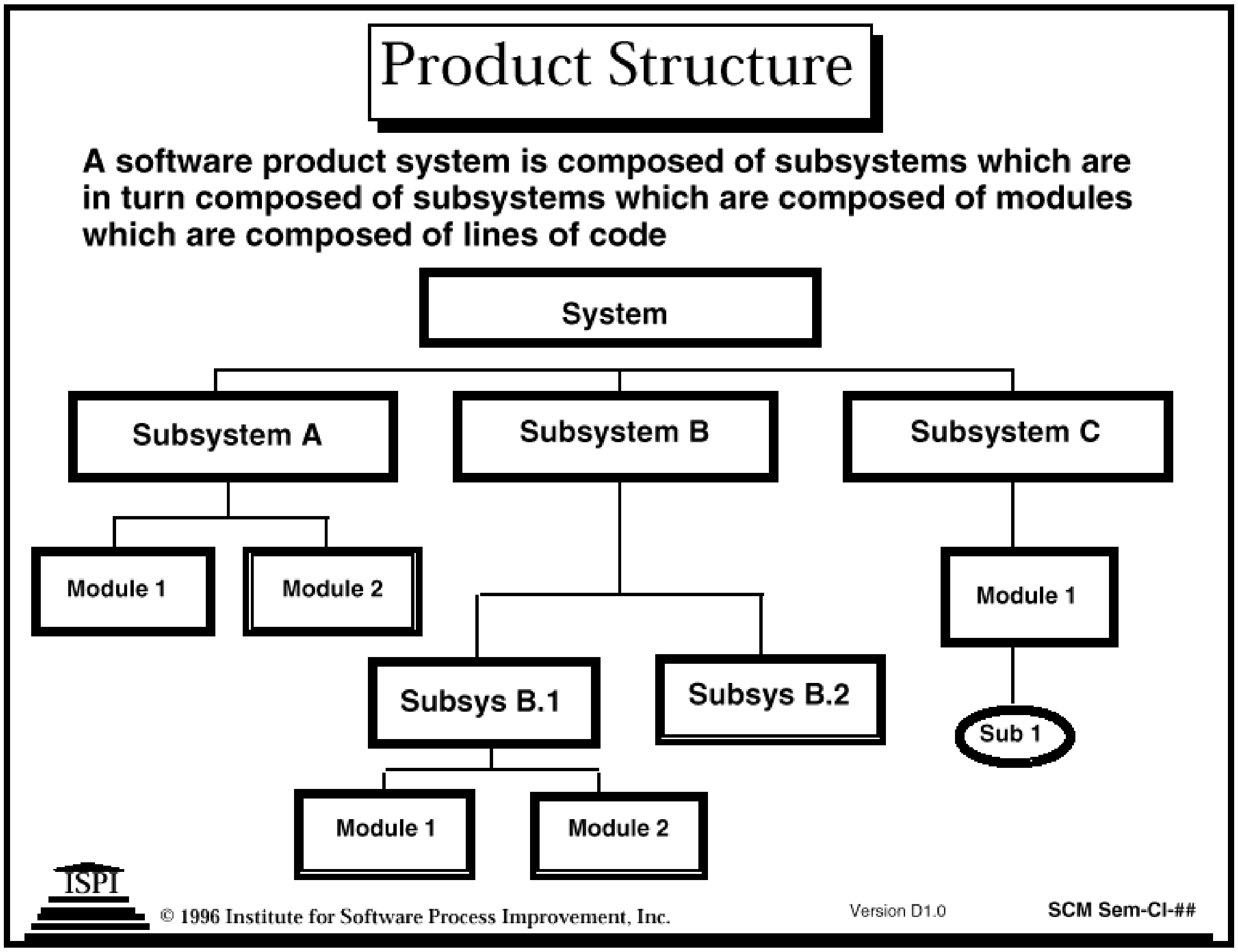
Product Structure
A software product system is composed of subsystems which are in turn composed of subsystems which are composed of modules which are composed of lines of code
System
Subsystem A
Subsystem B
Subsystem C
Module 1
Module 2
Subsys B.1
Subsys B.2
Module 1
Sub 1
Module 1
Module 2
ISPI
© 1996 Institute for Software Process Improvement, Inc.
Version D1.0
SCM Sem-CI-##

Figure 2. Product Structure

Figure 2 is a simple example of a software product system composed of subsystems and modules. Each system or subsystem component may have associated with it an “Include” file for code or data and a “Make” file for creating compiled and linked systems or subsystems. A discussion between the project and the SCM Representative can help the Project Leader look critically at the software architecture and plan for evolutionary builds that can be controlled and tested at the developmental and system level.

Baselining

Change is a fact of life in software development. Customers want to modify requirements. Developers want to modify the technical approach. Management wants to modify the project approach. Modification is necessary, because, as time passes, all parties know more about what they need, which approaches would be best, and how to get it done and still make money. The additional knowledge becomes the driving force behind most changes.

The fundamental success of any development effort is dependent on well-defined reference points against which to specify requirements, formulate a design, and specify changes to these requirements and the resultant designs. The term baseline is normally used to denote such a reference point. A baseline is an approved snapshot of the system at appropriate points in the development lifecycle. A baseline establishes a formal base for defining subsequent change. Without this line or reference point, the notion of change is meaningless. A baseline could be:

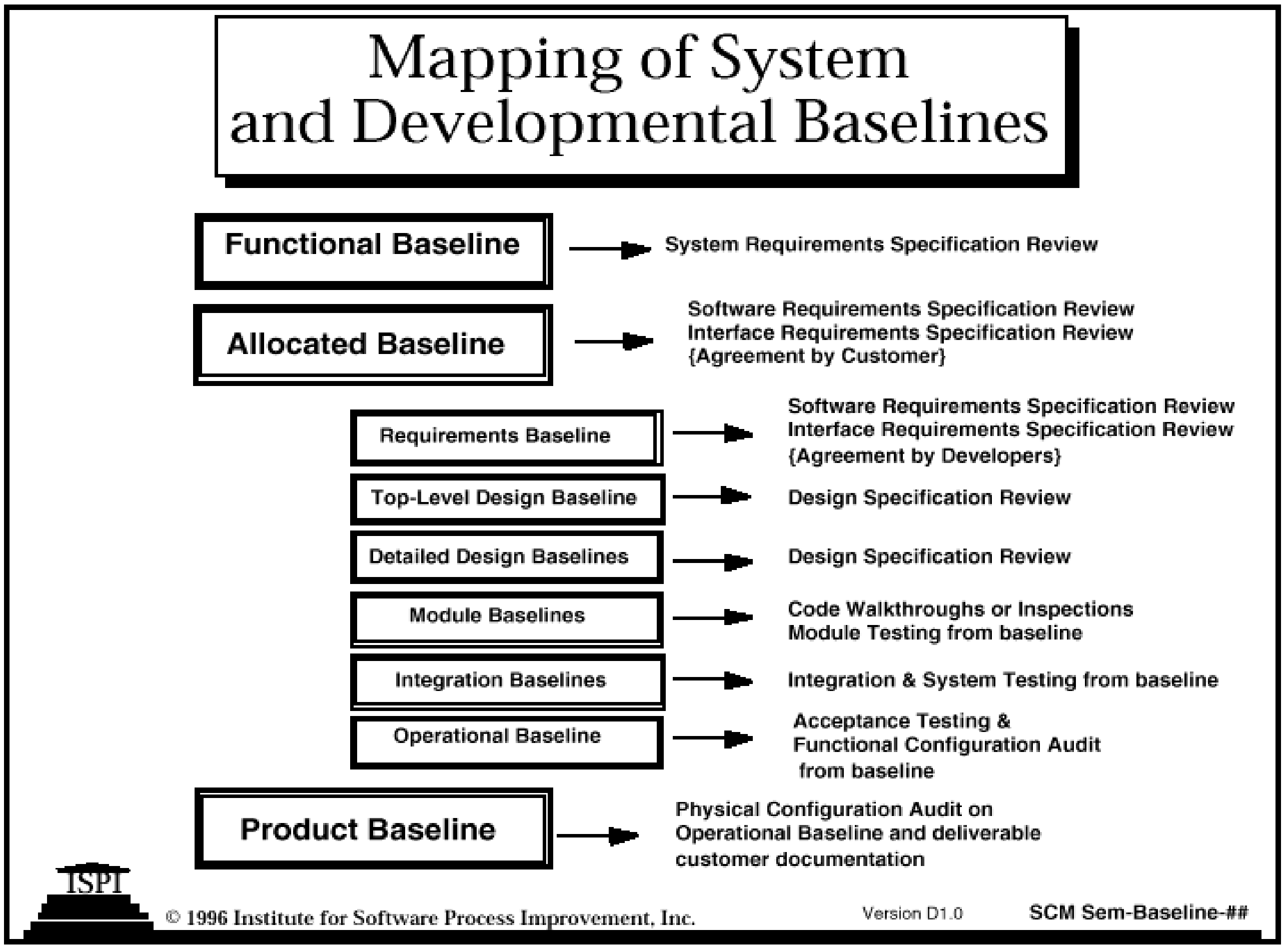
Mapping of System and Developmental Baselines
Functional Baseline
System Requirements Specification Review
Allocated Baseline
Software Requirements Specification Review
Interface Requirements Specification Review
{Agreement by Customer}
Requirements Baseline
Software Requirements Specification Review
Interface Requirements Specification Review
{Agreement by Developers}
Top-Level Design Baseline
Design Specification Review
Detailed Design Baselines
Design Specification Review
Module Baselines
Code Walkthroughs or Inspections
Module Testing from baseline
Integration Baselines
Integration & System Testing from baseline
Operational Baseline
Acceptance Testing &
Functional Configuration Audit
from baseline
Product Baseline
Physical Configuration Audit on
Operational Baseline and deliverable
customer documentation
ISPI
© 1996 Institute for Software Process Improvement, Inc.
Version D1.0
SCM Sem-Baseline-##

Figure 3. Mapping of System and Developmental Baselines

- A specification (e.g., requirements specification, design specification)
- A product that has been formally reviewed and agreed upon
- A partial system

A baseline is a record of a contract. It serves as the basis for further development. It should be changed only through an agreed upon change procedure. A baseline helps a project to control change without seriously impeding justifiable change. A baseline will help a project to control the identified configuration items but not constrain early development excessively from the aspects of time, money, or resources. Before a baseline is established, change may be made quickly and informally. Once a baseline is established, change can be made but a specific, formal procedure must be applied to evaluate and verify each change. The items in the baseline are the basis for the work in the next phase of the software development cycle. The items of the next baseline are measured and verified against previous baselines before they become baselines themselves.

Figure 3 illustrates both the types of baselines that are typical and the quality functions that may be used to ensure that the work products are of the highest quality before they are baselined. Functional Baseline, Allocated Baseline, and Product Baseline are most often thought of as Organizational or System Baselines. Requirements Baseline, Design Baselines, Module Baselines, Integration Baseline (Component & System), and Operational Baseline are often thought of as Project or Developmental Baselines. System baselines are the records of contract made with the external customer. Developmental baselines are agreements to ensure the product integrity as it moves from phase to phase.

Configuration Control

In the ideal world, once a configuration item is fully approved there would be no need to change. In the real world, new versions of a configuration item are needed for a variety of reasons:

- The requirements for the system change
- The boundaries between items in the design hierarchy changes
- The specification of an item is incomplete or wrongly interpreted
- An error is found that was not detected during the configuration items review
- The software environment changes in a way that necessitates change

In each case, a new version of a configuration item is needed which supersedes the earlier version. Without change control, a software engineer could make an important change to a configuration item or its interfaces without a lot of extra work and red tape. However, no record would be kept of what the change was, why the change was requested, who approved the change, who made the change, and who verified the change. In addition, it would be hard to find out:

- "Why doesn't my software link this morning? It linked last night!"
- "Why does this regression test fail now? It worked yesterday!"
- "Why does the product behave this way not? It didn't before!"
- "Why are we running out of memory now? We did not have that problem yesterday!"

All changes made to the configuration management baselines or baselined software configuration items should be done according to a documented change control process. The change control process should specify:

- Who can initiate the change requests
- What the criteria is for placing the software components under formal change control
- The "change impact" analysis expected for each requested change
- How revision history should be kept
- The Check-in/Check-out procedures
- The process the Software Configuration Control Board follows to approve changes
- How change requests will be linked to the Trouble Reporting System
- How change requests are tracked and resolved
- The reviews and /or regression tests that must be performed to ensure that changes have not caused unintended effects on the baseline
- The procedure that will be followed to update all affected software life-cycle components to reflect the approved changes.

In summary, to have effective configuration control that truly supports project development, it is important to establish a change control process that specifies: Who can initiate the change requests; the individuals, group, or groups who are responsible for evaluating, accepting, and tracking the change proposals for the various baselined products; the "change impact" analysis expected for each requested change; and how the change history should be kept.

To control the organizational or system baselines or contracts with the external customers, many organizations establish one or more Software Configuration Control Boards (SCCB). The SCCB is to ensure that every change is properly considered and coordinated. This SCCB may include members from Program Management, Systems Engineering, Software Engineering, Software Quality Assurance, Software Configuration Management, Independent Test, Documentation, Hardware Engineering and even may include a customer representative. They would be responsible for receiving and initially evaluating the change requests that come from all sources (i.e., customer, engineering, marketing, trouble reports, program management) and performing triage to get the most critical or significant change requests to the right people for impact analysis. Following the impact analysis, the SCCB would ensure that all affected groups were able to recommit to the new requirements. The SCCB could make the decision to implement the change request, defer it to the next release, or discard it altogether. It is also possible that the SCCB would have to seek additional information before a decision could be made.

Once the Allocated Baseline was created and the customer had accepted the Software Requirements Specification and Interface Specification, the control of the work products and system components would come under Developmental Configuration Control. Normally this means that decisions to make changes would be decided by the Project Leader. If a change to a software module resulted in a required change to an interface module to a hardware device, the Project Leader might share the approval responsibility with the appropriate Hardware Manager. When the software passes to the Integration and Test Stage, the Project Leader may share approval authority to make changes with the Integration and Test Manager. When the product is ready to be shipped to the customer and a Product Baseline is established, the SCCB again becomes the approval authority. What baselines, when they are created during the software lifecycle, and who the approval authority or authorities is/are becomes a part of each project's Software Configuration Management Plan.

Configuration Management Status Accounting

Configuration Management Status Accounting involves maintaining a continuous record of the status and history of all baselined items and proposed changes to them. It includes reports of the traceability of all changes to the baseline throughout the software lifecycle and it should answer the questions: What changes have been made to the system? and What changes remain to be implemented? Configuration Management Status Accounting provides visibility into the system evolution by recording and reporting the status of all items and the status of all requests for change. Questions that Configuration Management Status Accounting should be able to answer include:

- What is the status of an item? A programmer may want to know whether a specification has been fully approved. A programmer may want to know whether a subsystem has been tested so that the programmer can test his modules which interface with that subsystem. A project leader will wish to track the progress of a project as items are developed, reviewed, tested, and integrated.
- Has a change request been approved or rejected by the SCCB?
- Which version of an item implements an approved change request? Once a requested enhancement of a library routine is implemented, the originator and other developers will want to know which version of the routine contains the enhancement.
- What is different about a new version of a system? A new version of a software system should be accompanied by a document listing the changes from the previous version. The change list should include both enhancements and fixes to faults. Any faults that have not been fixed should also be named and described.
- How many faults are detected each month and how many are fixed? Faults are continuously detected during the operational use of the system. Comparing the number of detected and fixed faults helps to assess the stability of the latest release of the system. Tracking the number of faults also helps the Program Manager to decide when to make a new release of the system.
- What is the cause of the trouble report? Trouble reports can be categorized by their causes: violation of programming standards, inadequate user interface, or left out customer require-

ments. Sometimes when it is discovered that many faults have a similar cause, action can be taken to improve the process and stop such faults from recurring.

Configuration Management Status Accounting is the means by which key project or system information can be communicated to everyone. Project members can easily determine which configuration item should be used, whether it is subject to a change request, and what a build consists of. Project Leaders can easily determine what configuration items passed review, which changes have been completed, which changes are still in progress, how many changes have been accepted, which modules are the most volatile, what a build consists of, and what has been delayed to the next release.

Interface Control

The definition of interfaces is one of the most important Software Configuration Management planning and tracking activities. There must be common agreement of each group or organization's responsibility. Any proposed changes to the product or baselined configuration items can be considered and evaluated by all affected groups.

There are two basic types of interfaces that must be considered: Organizational interfaces and Technical interfaces. Organizational interfaces are those which Configuration Management controls the transfer of configuration items from vendor to customer, from project to project, and from co-developer to co-developer. SCM ensures that the correct configuration items are sent to the correct people. Organizational interfaces also include life-cycle phase interfaces. Phase interfaces become critical when control of the product is being transitioned between different groups (e.g., software development group to independent test group for formal testing). Technical interfaces are descriptions that should be placed under configuration management control like any other configuration item. Technical interfaces include system, user, software, hardware, and communication interfaces.

Subcontractor Control

If a portion of a software development project is to be subcontracted to another organization, the responsibility for the software configuration management generally belongs to the contracting organization. The subcontractor is normally only responsible for the portion of the work that his/her organization is tasked to perform. The integration of the subcontracted work is normally the responsibility of the organization who subcontracted portions of the work An effective SCM system greatly increases the opportunity to have portions of the product subcontracted out and then integrated back into a whole that satisfies the customer's technical and quality requirements. SCM must be applied to a subcontractor in order to ensure that the subcontractor is able to maintain the integrity of the subsystem for which it has contracted. This includes placing necessary life-cycle products under configuration control to ensure consistency with the main development effort and maintaining a subcontractor's software library that will release the agreed upon configuration items or subsystems to the contracting organization.

Software Configuration Audits

Configuration auditing verifies that the software product is built according to the requirements, standards, or contractual agreement. Auditing also verifies that all software products have been produced, correctly identified and described and that all change requests have been resolved. A software configuration audit should periodically be performed to ensure that the SCM practices and procedures are rigorously followed. The integrity of the software baselines must be assessed. The completeness and correctness of the software baseline library contents must be verified. The accuracy of the implementation of the changes to the baselines must be verified to ensure that the changes were implemented as intended. It is recommended that a software configuration audit be performed before every major baseline change.

Software configuration auditing should be continuous, with increased frequency and depth throughout the lifecycle. Types of configuration audits include Functional Configuration Audits, Physical Configuration Audits, In-Process Audits, and Traceability Audits.

Software Library

The Software Library should contain the items that are important to a software project including source code, user documentation, system documentation, test data, support software, specifications, and project plans. The SCM Library typically stores the configuration items and prevents unauthorized changes to the baselined items. The library system should:

- Support multiple control levels of SCM
- Provide for the storage and retrieval of configuration items
- Provide for the sharing and transfer of configuration items among affected groups and among control levels within the library
- Provide for the storage and recovery of archive versions of configuration items
- Provide service functions such as checking status, verifying the presence of all built items, and integrating changes into a new baseline
- Ensure the correct creation of products from the software baseline library
- Provide for the storage, update, and retrieval of SCM records
- Support the production of SCM reports
- Support the tracing of requirements, forwards and backwards, throughout the lifecycle

A software library provides safe and secure storage for configuration items (key project components) so that they cannot be changed without authorization.

Acceptance of items (new or revised) into the library is strictly controlled to ensure that everyone accessing items in the library can have complete confidence in their integrity. A number of different libraries may be established to hold different types of items or to provide different levels of control.

Software Configuration Management Plan

The Software Configuration Management Plan is the document which describes how a project will manage configurations. The SCM Plan should cover:

- Scope of the Plan including the project, the software to be developed, and the life-cycle phases
- The relationship between the SCM plan and the other standards or plans which describe how the project will be managed (e.g., Software Development, SQA Plan)
- SCM Roles and Responsibilities
- Configuration Identification
- Baselining
- Configuration Control
- Configuration Management Status Accounting
- Interface Control
- Subcontractor Control
- Software Configuration Audits
- Software Library

Basic Configuration Management

Even if an organization has little or no configuration management in place, five very simple steps will add a great deal of control and project tracking information: 1) Formalize the use of reviews before a configuration item is baselined, 2) Uniquely identify system components, 3) Establish simple change control, 4) Build up a repository of configuration items, change requests, and problem reports, and 5) Restrict access to the project library.

Summary

Software Configuration Management is one of the most important process improvement tools that a project leader can use to evolve and deliver his/her product in a controlled manner.

References

[1] Babich, Wayne, Software Configuration Management , Addison-Wesley Publishing Company, 1986

[2] Bersoff, Edward H., Henderson, Vilas D., Siegel, Stanley G., Software Configuration Management , Prentice-Hall, 1980

[3] Buckley, Fletcher J. Implementing Configuration Management, Computer Society Press, 1996.

[4] Bryan, William L. and Siegel, Stanley G. Software Product Assurance, Elsevier Science Publishing Co, 1988.

[5] Humphrey, W. Managing the Software Process, Addison-Wesley, 1990.

[6] IEEE, IEEE Software Engineering Standards Collection, IEEE Press, 1994.

[7] ISO, ISO 9000 Quality Management, ISO Standards Compendium, 1994.

[8] Kasse, Tim, “Project Leader Exploratory Question Database”, Institute for Software Process Improvement, 1995.

[9] McDermid, John, Software Engineer’s Reference Book, CRC Press, 1994

[10] Paulk, Mark C., Curtis, Bill, Chrissis, Mary Beth, and Weber, Charles V., “Capability Maturity Model for Software, Version 1.1" Software Engineering Institute, CMU/SEI-93-TR-24, February 1993.

[11] Paulk, Mark C., Weber, Charles V., Garcia, Suzanne M., Chrissis, Mary Beth, and Bush, Marilyn, “Key Practices of the Capability Maturity Model Version 1.1,” Software Engineering Institute, CMU/SEI-93-TR-25, February 1993.

[12] Pressman, Roger S., Software Engineering , McGraw Hill, 1987

[13] Rigby, Ken, Configuration Management Plan , World Wide Web, 1996

[14] Schulmeyer, G. Gordon, McManus, James I. (Editors), Handbook of Software Quality Assurance , Van Nostrand Reinhold Company, 1987.

[15] STSC, Software Configuration Management Technology Report , Software Technology Support Center, 1994

[16] Walker, Gary, “What is Configuration Management?”, Alcatel Alsthom internal presentation, 1996

[17] Whitgift, David, Methods and Tools for Software Configuration Management , John Wiley & Sons, 1991.